Nazca: Detecting Malware Distribution in Large-Scale Networks

Luca Invernizzi
UC Santa Barbara
invernizzi@cs.ucsb.edu

Stanislav Miskovic
Narus, Inc.
smiskovic@narus.com

Ruben Torres
Narus, Inc.
rtorres@narus.com

Sabyasachi Saha
Narus, Inc.
ssaha@narus.com

Sung-Ju Lee
Narus, Inc.
sjlee@narus.com

Marco Mellia
Politecnico di Torino
mellia@polito.it

Christopher Kruegel
UC Santa Barbara
chris@cs.ucsb.edu

Giovanni Vigna
UC Santa Barbara
vigna@cs.ucsb.edu

Abstract—**Malware remains one of the most significant security threats on the Internet. Antivirus solutions and blacklists, the main weapons of defense against these attacks, have only been (partially) successful. One reason is that cyber-criminals take active steps to bypass defenses, for example, by distributing constantly changing (obfuscated) variants of their malware programs, and by quickly churning through domains and IP addresses that are used for distributing exploit code and botnet commands.**

We analyze one of the core tasks that malware authors have to achieve to be successful: They must distribute and install malware programs onto as many victim machines as possible. A main vector to accomplish this is through drive-by download attacks where victims are lured onto web pages that launch exploits against the users' web browsers and their components. Once an exploit is successful, the injected shellcode automatically downloads and launches the malware program. While a significant amount of previous work has focused on detecting the drive-by exploit step and the subsequent network traffic produced by malware programs, little attention has been paid to the intermediate step where *the malware binary is downloaded*.

In this paper, we study how clients in real-world networks download and install malware, and present Nazca, a system that detects infections in large scale networks. Nazca does not operate on individual connections, nor looks at properties of the downloaded programs or the reputation of the servers hosting them. Instead, it looks at the telltale signs of the malicious network infrastructures that orchestrate these malware installation that become apparent when looking at the collective traffic produced and becomes apparent when looking at the collective traffic produced by many users in a large network. Being content agnostic, Nazca does not suffer from coverage gaps in reputation databases (blacklists), and is not susceptible to code obfuscation. We have run Nazca on seven days of traffic from a large Internet Service Provider, where it has detected previously-unseen malware with very low false positive rates.

NDSS '14, 23-26 February 2014, San Diego, CA, USA
Copyright 2014 Internet Society, ISBN 1-891562-35-5
http://dx.doi.org/10.14722/ndss.2014.23269

I. Introduction

Malware is one of the most severe security threats on the Internet. Once infected with malicious code, victim machines become platforms to send email spam messages, launch denial-of-service attacks, and steal sensitive user data.

A key challenge for attackers is to install their malware programs on as many victim machines as possible. One approach is to rely on social engineering: for example, attackers might send email messages that entice users to install attached malware programs. While this technique works, it requires the cooperation of victim users, and hence is often ineffective. An alternative, and more effective, approach is to lure users onto web pages that launch exploits against vulnerabilities in web browsers (or their components, such as the PDF reader or the Flash player). In this case, no user interactions are required, and the malware is surreptitiously installed and launched on the victim's machine. The effectiveness and stealthiness of drive-by downloads have made them the preferred vehicle for attackers to spread their malware, and they are the focus of the work presented in this paper.

The infection process in a drive-by download attack can be divided into three phases. During the first phase (the *exploitation phase*), the goal of the attacker is to run a small snippet of code (shellcode) on the victim's host. To this end, the attacker first prepares a website with drive-by download exploit code. When a victim visits a malicious page, the browser fetches and executes the drive-by code. When the exploit is successful, it forces the browser to execute the injected shellcode. In the subsequent second phase (the *installation phase*), the shellcode downloads the actual malware binary and launches it. Once the malware program is running, during the third phase (the *control phase*), it unfolds its malicious activity. Typically, the malware connects back to a remote command and control (C&C) server. This connection is used by attackers to issue commands, to "drop" new executables onto the infected host to enhance the malware's functionality, and receive stolen data.

Most current techniques protecting users against malware focus on the first and the third phases. A large body of work targets the initial exploitation phase, trying to detect pages that contain drive-by download exploits and prevent browsers from visiting a malicious page in the first place. For example, honeyclients crawl the web to quickly find pages with exploit code, and turn these findings into domain and URL blacklists. Attackers have responded by quickly rotating through malicious domains, making blacklists perpetually out-of-date. Also, attackers have started to aggressively fingerprint honeyclients and obfuscate their code to avoid detection [1].

Another large body of work focuses on the control phase, attempting to identify the execution of malicious code on the end host. Antivirus (AV) programs are probably the main line of defense that is employed by most users. AV software relies mostly on signatures to detect malicious programs when they are stored to disk or executed. Unfortunately, these programs are seeing diminishing returns in successful detections [2], as malware writers modify their programs to avoid detection [3]. Researchers have also developed solutions that use signatures or reputation-based systems to identify and block the directives that the malware distributor issues to the infected hosts after a successful infection. Attackers have reacted by encrypting their communication channels and even using steganographic techniques to make their command and control (C&C) traffic appear legitimate.

So far, little attention has been paid to the installation phase. At first glance, this makes sense as in the installation phase, the shellcode typically issues an HTTP request that fetches a program from a remote server, and then installs and executes the malware locally. Often, this request is done by simply invoking available functions in the user's browser. From the network point of view, such connections are hardly suspicious, and look essentially identical to legitimate requests performed by users who download benign programs (e.g., updates or shareware programs).

However, the situation changes significantly when "zooming out" and leaving the myopic view of individual malware downloads. Instead, when considering many malware downloads together – performed by different hosts, but related to a single campaign – a malware distribution infrastructure becomes visible. In some sense, this malware distribution infrastructure acts like a content distribution network. However, there are also differences, and these differences can be leveraged to identify cases where malicious content (malware programs) are distributed.

We present Nazca, a system that aims to detect web requests that are used to download malware binaries. Similar to the drawings in the Nazca desert, the malware downloads (and the supporting distribution infrastructure) becomes more apparent when observing a larger part of the picture/network. Our system detects these large-scale traits of malware distribution networks. Thus, it is designed to operate in large-scale networks, such as Internet Service Providers (ISPs), large enterprise networks, and universities. Moreover, our system focuses on malware downloaded through HTTP requests. This design choice is driven by the observation that an overwhelming majority of drive-by exploits use the web to download malware binaries. Finally, Nazca does not perform any analysis of the content of web downloads, except for extracting their MIME type. That is, we do not apply any signatures to the network payload, do not look at features of the downloaded programs, and do not consider the reputation of the programs' sources. This allows us to identify the downloads of previously-unseen malicious software, enabling zero-day malware detection.

Our system monitors web traffic between a set of hosts (typically, machines in the protected network) and the Internet. The goal is to identify the connections that are related to malware downloads. To this end, Nazca operates in three steps: In the first step, the system identifies HTTP requests and extracts metadata for subsequent analysis. This metadata includes information on the connection endpoints, the URIs, and whether an executable program is being downloaded.

In the second step, Nazca identifies suspicious web connections whose HTTP requests download an executable file. There are certain properties associated with the connection that make it appear different from legitimate downloads. These properties are designed to capture techniques employed by malware authors to hide their operations from traditional defense systems. Such evasive techniques include domain fluxing, malware repackaging, and the use of malware droppers for multi-step installations. An interesting and favorable trait of our approach is that it complements well to the existing detection mechanisms. That is, when malware authors employ techniques to evade traditional approaches (such as malware signatures or IP/domain reputation systems), their downloads are easier to recognize for Nazca.

In the third step, Nazca aggregates the previously-identified suspicious connections (candidates). The goal is to find related, malicious activity so as to reduce potential false positives and focus the attention on the most significant infection events. Here, we build a graph of the malicious activities that Nazca detected. This enables the reconstruction and observation of entire malware distribution networks.

We have evaluated Nazca's effectiveness on a one-week traffic dataset from a commercial ISP. Our results show that Nazca is effective in detecting malicious web downloads in large-scale, real-world networks.

The main contributions of this paper are as follows:

- We present a novel approach to identify web requests related to malware downloads and installations. Our system operates in large networks, and identifies traits of orchestrated malware distribution networks. Since our approach does not analyze the downloaded programs, Nazca can identify unseen-before malware, and is not defeated by content obfuscation.
- We introduce a three-step process to detect malware downloads. First, we extract a short summary of HTTP requests metadata. We then identify suspicious candidate connections that exhibit features that are anomalous for benign downloads. These features capture evasive attempts of malware authors to avoid detection by traditional defense systems. Last, we combine candidates to identify clusters of malicious activity.
- Using seven days of traffic from a commercial ISP, we evaluate the effectiveness of our system to detect malware downloads, and we compare this effectiveness to popular blacklists.

II. Approach

Nazca aims at identifying HTTP connections downloading malicious software. Our system is designed to operate in large-scale networks, such as ISPs, large enterprise networks, and universities. That is, we assume that our system is positioned at a vantage point where it can observe the traffic between a large set of hosts and the Internet. In Section VII-D, we explore in more detail how many hosts Nazca needs to observe to be effective.

The ability to monitor the web traffic from larger number of machines provides the key advantage that our system can see – and correlate – multiple malware downloads related to a specific campaign. This is crucial, since we do not analyze the downloaded binaries or take into account the reputation of the source of the download. Attackers have significant freedom in crafting their malware programs, and have long used packers and code obfuscation to limit the utility of traditional AV signatures. Malware authors can also use different IPs and domains to serve their software. As a result, blacklists and domain reputation systems naturally lag behind and suffer from limited coverage. By ignoring features that attackers can easily change, and instead focusing on properties of the malware distribution infrastructure, Nazca has the ability to detect previously-unseen, malicious code.

Nazca inspects only IP and TCP packets headers, HTTP headers, and a small portion of HTTP responses. We limit Nazca's analysis to HTTP traffic because we observed that it is the protocol of choice for the vast majority of malware in current circulations (see Section VII-A for details). One reason for distributing malware via HTTP is that it is typically allowed through corporate firewalls. When attackers implement custom protocols to download their binaries, the risk is higher that application-aware firewalls would simply drop the traffic (and hence, block the infection).

The payload analysis of an HTTP response is limited to detecting the MIME type of the content that the web server returns and calculating a hash of (the beginning of) it. We need to determine the MIME type of the server response to distinguish between program (binary) downloads and other, irrelevant content (such as web pages, images, etc.). We compute the hash over the first portion of a program download to determine whether two programs are (likely) identical or different. Since we do not inspect properties of the code that is being downloaded, but instead focus solely on the malware distribution infrastructure, Nazca has the ability to detect previously-unseen (zero-day) malware programs.

The following sections discuss the main steps of Nazca in more detail.

III. Extraction

During the extraction step, Nazca analyzes network traffic by recording live packets on the wire or reading the PCAP files. The goal of this step is to extract a metadata record for each connection of interest. Nazca extracts a record for each HTTP connection that initiates a download of a file whose MIME type is not in our whitelist (MIME types of popular formats for images, videos, audio, and other innocuous files).

More specifically, Nazca reassembles packets and aggregates them into streams. For performance reasons, we do not reassemble the complete stream, but just enough to perform protocol detection. We discard any stream that does not contain HTTP requests or responses. Whenever we find an HTTP request, we analyze the corresponding answer from the server. In particular, we are interested in determining the MIME type of the object that the server returns. To this end, we do not trust the server HTTP `Content-Type` header, as we have found it to be often misleading. Instead, we perform a quick `Content-Type` sniffing with `libmagick` (the library that powers the `file` command in GNU/Linux). We also attempt to decompress payloads zipped with popular protocols (such as zip, gzip, bzip2, etc.).

Whenever our analysis recognizes that a downloaded file's MIME type is not whitelisted, we record the following information: source (client) and destination (server) IP address and port, URI that the client requests (everything, including parameters), the value of the `User-Agent` HTTP header field, and a content hash of the uncompressed first k bytes at the beginning of the file. k is a configurable value that should be large enough to minimize collisions, without sacrificing performance (see [4]). In this paper, we use the first kilobyte.

The records that correspond to interesting connections can be recorded in real-time, as soon as the information is available. In a typical network, the amount of data that is collected in this fashion is very small (compared to the data volume on the wire). Hence, we do not expect any scalability issues, even for very large networks. For example, in Section VII-E, we show that we need to store only 286 KB of metadata per client per day. Since our default time window is typically less than a day, old data can be discarded.

IV. Candidate Selection

Nazca works by correlating information from multiple connections. In the candidate selection step, Nazca considers the set of all metadata records that are captured during a certain time period T. One can run the candidate selection step periodically, analyzing the connections in chunks of T. Alternatively, one could keep a sliding window of length T. The goal of this step is to produce a candidate set that contains suspicious connections. These candidates are connections that exhibit behavior that is typically associated with malicious download or install activity.

We use four different techniques to identify candidates. These four techniques focus on cases of repackaged malware, distributed malware hosting, ad hoc malicious domains, and the use of malware droppers. Our techniques are designed to be fast so that they can work on large-scale datasets. Of course, we cannot expect a single technique to handle all ways in which malware is distributed. However, our experiments demonstrate that the combination of our techniques provide good coverage. Moreover, if needed, it is easily possible to add additional techniques.

We now present the techniques used in the filtering step in detail. Keep Figure 1 for reference on how to interpret the various symbols on the graphs that accompany them.

A. Detection of File Mutations

Our first detection mechanism captures attempts by malware authors to bypass antivirus signatures. To avoid signature-based detection at the end host, malware authors frequently

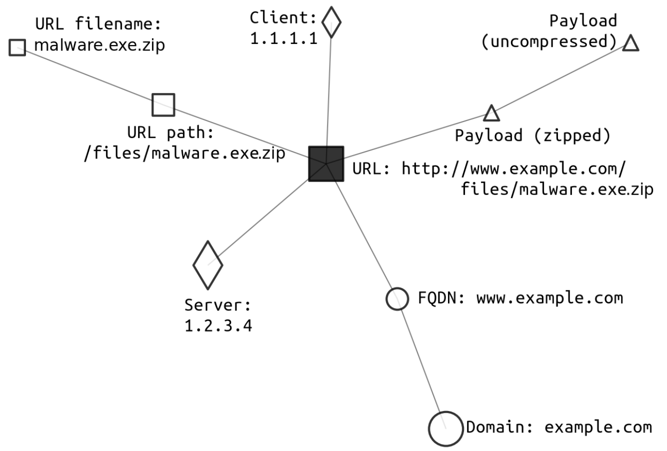


Fig. 1. Legend to interpret graphs throughout the paper.

change their files. Typically, this is achieved by packing the basic malware program, using a different (encryption) key for each iteration. The technique of preparing and serving a new variant of a malware program (or family) is called server-side polymorphism. Alternatively, malware authors can prepare a set of malicious executables, and serve a different one for each request.

Our technique to detect file mutations (i.e., server-side polymorphism) works by looking for download records that are (i) associated with a single URI and (ii) download more than n different files (determined by the hashes of the files). We discuss the choice of a suitable threshold for n later. The nice property of this technique is that increased efforts by cyber-criminal to avoid being detected by antivirus signatures make it easier for our technique to detect such activity.

Examples of malicious domains that trigger this technique include cacaoweb.org, which hosts URIs that serve an executable that changes every few hours (and that is still active at the time of writing). Other examples are the three domains www.9530.ca, www.wgcqsf.com, and 585872.com. All three domains host the same URL path /dlq.rar, which contains malware that attacks Internet Explorer. This example is particularly interesting because it shows that the malware distributors are trying not only to evade antivirus signatures by repackaging their malware (therefore exposing to detection by our technique), but also to achieve robustness to blacklisting and take-downs by using multiple domains, a behavior that is handled by our subsequent techniques.

We expect that only a small minority of benign web sites exhibit a similar behaviour, where a URI keeps pointing to different executable files within a short time period. One reason for this is that quickly rotating files negatively affects browser and proxy caching, which in turn increases the page loading time and the server load. Another reason is that such behavior might cause unexpected timing effects, where the code that a client receives depends on the precise time of access. We found only one family of programs that seem to use such behavior, namely, antivirus software itself. Since there are only a few domains associated with well-known AV software, we simply whitelist those domains.

B. Detection of Distributed Hosting and Domain Fluxing

Cyber-criminals, like regular content providers, must ensure that their sites (and malware programs) are always available. Any downtime results in fewer compromised users, and thus, a loss of revenue. For this reason, many cyber-criminals replicate their infrastructure on multiple servers, much like Content Delivery Networks (CDNs). Unlike CDNs, cyber-criminals need to take one step further as they face defenders that are actively trying to shut them down. Thus, malicious distribution networks have different properties than benign CDNs.

Our technique works in two steps. First, we attempt to find CDNs. We then use a classifier to distinguish between legitimate and malicious CDNs. As we show later, we found that a decision tree classifier performs well for our purpose.

For the first step, we cluster domains so that two domains end up in the same cluster if they host at least one identical file (based on the file hash). That is, when we see web requests to URIs that link to a file with hash h on domains d_1 and d_2, respectively, both are put into the same cluster. We connect two clusters when we find a single file that has been hosted by at least one element (domain) in each cluster. All clusters that contain two or more elements are considered to be CDNs.

For the second step, we distinguish between malicious and benign clusters (CDNs). This distinction is made using a classifier that we trained on a small data set of manually labeled malicious and benign clusters. The classifier leverages six features, as described below:

Domain co-location: To reduce operating costs, some cyber-criminals opt to host different domain names on the same server (IP address), all serving the same malware (possibly, under different names). Benign software is less likely to be distributed in such manner, because a single host defeats the idea of redundancy and load balancing. This feature is equal to the number of hosts, divided by the number of domains.

Number of unique top-level domain names: To defeat domain blacklisting, malware typically uses a variety of top-level domains (TLDs). Legitimate CDNs might use several domains, but they are all sub-domains under a single TLD. For example, Facebook and Netflix use a single top-level domain to distribute their files. Exceptions exist in a form of legitimate software mirrors dispersed over different domains that commonly serve open software.

Number of matching URI paths and **Number of matching file names:** Legitimate CDNs are typically very well organized, and every server replicates the exact directory structure of the other ones. This is mainly for efficiency, which is the central focus of a CDN. Malicious distributed hosting instead focuses on avoiding blacklisting. Because of this, file names and paths are often different. For these features, we compute the number of URI paths (and filenames) that appear in more than one domain in the cluster, divided by the number of domains.

Number of URIs per domain: Malware distributors usually have a small number of URLs serving executables (often, a single one) from each of their domain. In contrast, legitimate CDNs have vast directory structures. We leverage

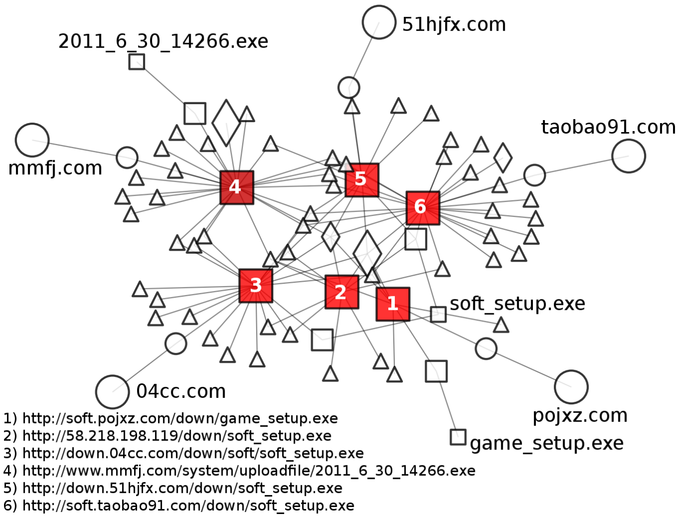

2011_6_30_14266.exe
51hjfx.com
taobao91.com
mmfj.com
4
5
6
3
2
1
soft_setup.exe
04cc.com
pojxz.com
game_setup.exe
1) http://soft.pojxz.com/down/game_setup.exe
2) http://58.218.198.119/down/soft_setup.exe
3) http://down.04cc.com/down/soft/soft_setup.exe
4) http://www.mmfj.com/system/uploadfile/2011_6_30_14266.exe
5) http://down.51hjfx.com/down/soft_setup.exe
6) http://soft.taobao91.com/down/soft_setup.exe

Fig. 2. Candidates selected by the Distributed Hosting technique.

this difference as an indication of maliciousness. We note that malware distributors, with some effort, could mimic this aspect of a legitimate CDN. However, we found this to be quite uncommon. For this feature, we count all the URIs across all domains in the cluster, and we divide this by the number of domains.

Served file types: File type diversity is also indicative: while most domains serve different file types, malware CDNs mostly serve executables. For this feature we divide the number of executables that we have seen hosted on the cluster, divided by the total number of files.

As an example of a malicious CDN, consider three URIs http://searchyoung.org/dfrg/dfrg, http://clickyawn.org/dfrg/dfrg, and http://clickawoke.org/dfrg/dfrg. We see that the same executable is offered on three different domains. The file is a fake optimization software called “HDD repair.” Here, all three domains offer the download under the same path `dfrg/dfrg`. This is not always the case, as shown in Figure 2. The malicious infrastructure in this figure is very interesting, as it showcases how distributed malware hosting tries to evade blacklisting by using multiple domains, servers, payloads, file names, and URL paths. We explain the details of this graph and the meaning of the different symbols in the next section. For this discussion, it can be seen how pairs of different URIs are linked together because they are linked to identical files.

Cyber-criminals often decide to quickly run through many domains, trying to stay ahead of domain blacklists. This is an evasion technique called *domain fluxing*. This evasive behavior is also captured by our technique as such activity looks similar to a CDN where only a single domain is used at any point in time.

C. Detection of Dedicated Malware Hosts

One technique that cyber criminals use to make their infrastructure more robust is to set up dedicated backend servers that deliver malware programs as part of a single domain. Typically, these servers only host the malicious binary. Traffic redirection services forward requests to the dedicated host as part of a successful drive-by download exploit. By hiding the dedicated malware server behind multiple layers of redirection, it is more difficult to detect and, ultimately, to take down. Since their sole purpose is delivering malware, dedicated hosts only host a small number of executable payloads (often times, only a single one) and at most a very small number of HTML and JavaScript files to support malware delivery. This is very different from a normal web server that hosts many different HTML pages, CSS files, and JavaScript programs.

With this technique, we check for those dedicated malware servers. To this end, we look for domains and IP addresses (in case the server is accessed directly via its IP address) that are involved in the download of a single executable file and host at most one other URI that can be an HTML page. From the set of candidates that this technique detects, we remove all instances that serve an executable that we have seen hosted on another domain (based on the file hash). The reason is that this technique specifically checks for single, dedicated hosts that are involved in a particular campaign. The previous technique already handles cases where malware authors distribute their samples over multiple hosts (domains).

As an example, we found dedicated servers (sub-domains) hosted under the `3322.org` domain. These servers were used by the Nitol botnet and have since been taken down after a legal action pursued by Microsoft [5]. Moreover, many executables that were found by this technique had names that try to lure the user to execute them (e.g., `emule_ultra_accelerator_free.exe`, `FreeTorrentViewer.exe`). These names are unique with respect to the rest of downloads that we observed.

D. Detection of Exploit/Download Hosts

When a vulnerable client visits an exploit website and the browser is successfully exploited, the shellcode silently downloads the second-step malware binary. In some cases, the host that serves the exploit is the same server that also delivers the malware program. Reusing the same server for both exploit and malware is a rational decision as the attacker knows that the server is up and unblocked. After all, it just delivered the infection.

Looking at the network traffic of such an incident, we observe two subsequent requests: one from the browser and one from the shellcode. If the malware writer is not careful and does not use the exploited browser's code to download the malware binary, the format of the second request will differ from a typical browser request from that user. With this technique, we leverage such discrepancies to identify suspicious requests. In particular, we look for destinations that are contacted more than once by the same IP address, but have different `User-Agent` HTTP header field values. Moreover, we check whether the `User-Agents` of the subsequent requests appear in requests that download the files. That is, we compute a score for each `User-Agent`, as the number of executables downloaded with that `User-Agent`, divided by the total number of downloads made with that agent (we count this during the Filtering Step). If this score is over a certain threshold (discussed in Section VI-D4), we consider the domain that is being contacted as a candidate for Nazca's last step. The assumption is that malicious download requests often use the same (hard-coded) `User-Agent` values. Hence, when the download request involves a `User-Agent` value that is indicative of download requests in general, the corresponding requests are considered suspicious. Otherwise, they are discarded as likely benign.

There are legitimate cases that could trigger this technique but result in false positives: a user could access the same site with multiple browsers (for example, apps in iOS devices have different `User-Agent` strings), or multiple users behind a NAT or a fast-rotating DHCP might access a specific site, and trigger our detection. We mitigate these issues by restricting the maximum time delta δ between a browser and a suspicious `User-Agent` visiting a domain. In our experiments, all malicious connections were issued within one minute of the infection. Another mitigation that we did not employ is filtering out popular domains that are constantly hit by multiple browsers.

An example of a domain identified via this technique is pacedownloader.com. This site is a file-sharing site that hosts bait software with hidden, malicious functionality, alongside clean software. A user in our dataset downloaded GZIP2 decompressor from the culprit domain. Unfortunately, this software contained malicious code. Specifically, after being launched, the Trojan (malware) component issued an HTTP request to the same domains to download additional, unwanted software (Funtool, a browser toolbar). This second request had the `User-Agent` field set to `NSISDL/1.2`. This is a framework by NullSoft, designed to ease the creation of installers. This framework is very popular among malware writers because it allows downloading additional software at runtime. Interestingly, this framework is abused so frequently that it is blocked by Kaspersky's AV software, and is detected by Snort rules published by EmergingThreats.

V. Detection

The end result of Nazca's candidate selection step is a collection of URIs that have exhibited some suspicious behavior, and thus, have been selected by one (or more) of our candidate selection techniques. These techniques are designed to efficiently filter the majority of benign traffic; however, they occasionally misclassify benign traffic as suspicious, as they miss contextual information to make a more accurate decision.

To remove false positives and focus the attention of the security administrator on large malware campaigns, we leverage the scale of our dataset: in particular, the fact that it often contains multiple clients' interactions with various portions of a malware distribution infrastructure (which might span multiple hosts and domains). If enough of these interactions are captured, they expose to Nazca the overall structure of these malware distributions services and their interconnections. We call this a "malicious neighborhood"; that is, a collection of domains, hosts, executable files, and URLs that, bundled together, define and compose one (or multiple) malicious distribution services.

An example of a complete malicious neighborhood is given in Figure 3. Here, we have several clients that download several malicious executables from a variety of sources, hosted on several servers. The malware they are downloading turns them into a botnet's zombies. To receive commands from the botnet's C&C server, these clients connect to a variety of IP addresses, and download their instructions. Note that the overall structure of the malicious infrastructure is not visible when we operate on a single connection or single host level.

Why are malware distribution infrastructure interconnected? We see three fundamental reasons: cost effectiveness, availability, and business deals among cyber-criminals.

As any web developer, cyber-criminals face the need of maximizing their availability while limiting their expenses. Their resources are therefore limited: servers and domain registrations come with a price tag and need to be used efficiently, so that profits are maximized. Hence, they co-locate multiple malicious domains on the same server (e.g., Figure 2), and multiple malware specimens on the same domain. Moreover, to achieve availability against blacklisting, miscreants deploy multiple copies of their C&C servers and initial attack domains, and discard them as they get blacklisted.

Finally, the economic ecosystem supporting malware is complex, and cyber-criminals tend to specialize in offering a particular service [6]. For example, some focus on compromising hosts and selling their bots. Others purchase access to bots and use them as platform to send spam. Yet others develop exploits that are sold to those that want to infect machines. From the point of view of the malicious neighborhood graph, we expect to see clients that contact domains related to attackers that infect machines. Afterwards, we see infected machines connect to a different set of domains; those that are related to the C&C infrastructure of the malware that has been installed.

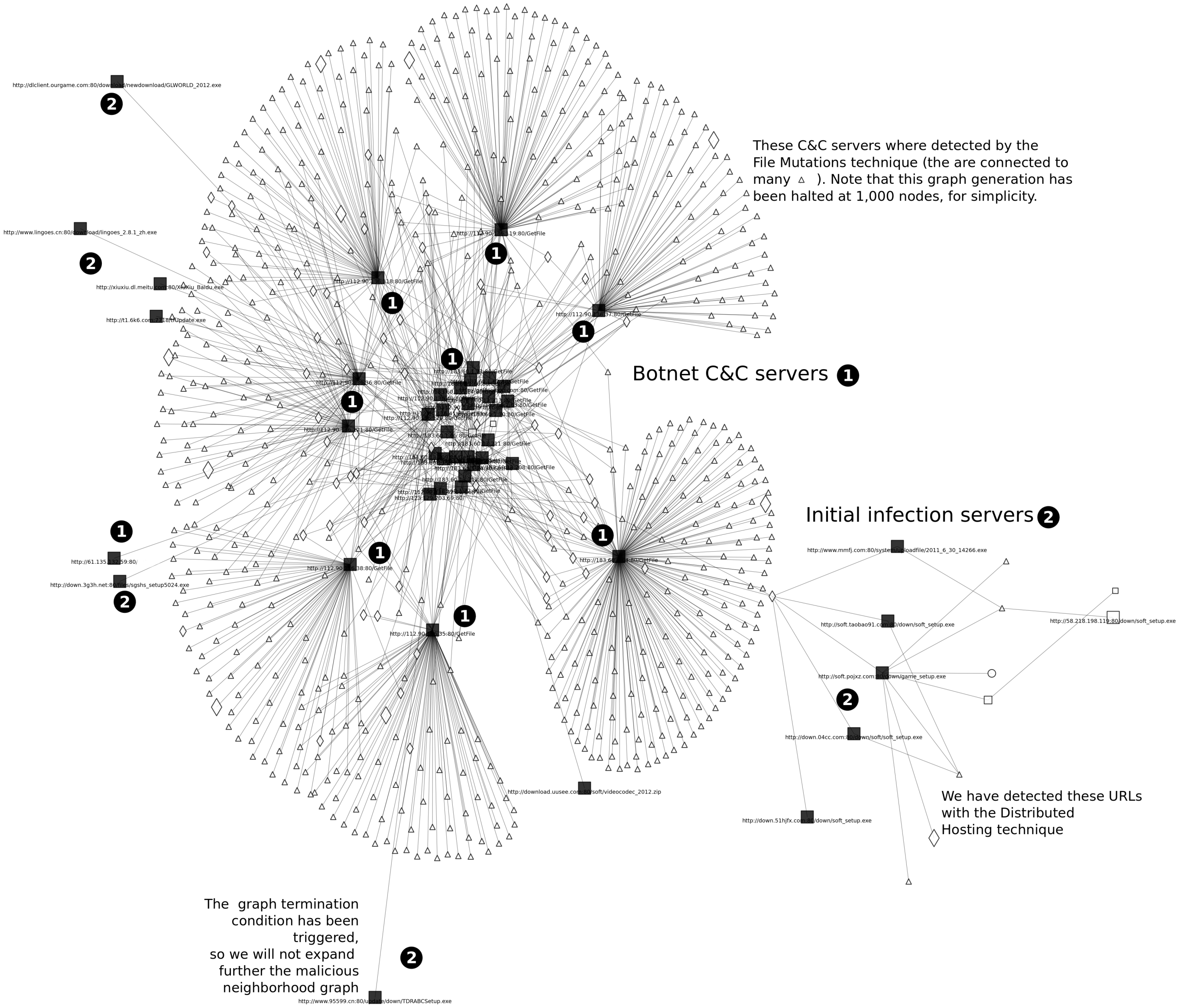
In this detection step, we build malicious neighborhood graphs for each candidates produced by the candidate selection step. If a candidate is actually malicious, it is likely (as we have experimentally validated) that there are other malicious candidates in the same graph, belonging to the same malicious neighborhood. From this graph, we then compute for each candidate a confidence score on its likelihood to be malicious. This score is based on how many other candidates appear in the same graph and how close they are to the input candidate in the graph.

A. *Malicious Neighborhood Graph Generation*

We define the malicious neighborhood as the collection of malicious activities related to a suspicious candidate, which is used as the starting point of the graph.

Neighborhood graphs are undirected and contain heterogeneous nodes that represent the following entities: IP addresses, domain names, FQDNs, URLs, URL paths, file names, and downloaded files (identified by their hash value). A simple graph that we invite the reader to keep as a reference is shown in Figure 1. It represents a zipped executable file being downloaded from `http://www.example.com/files-/malware.zip`.

We build these graphs incrementally. At initialization, the graph comprises only the seeding suspicious candidate; it can be a URL or a FQDN, depending on the technique that generated it. We then execute a series of iterations to grow



These C&C servers where detected by the File Mutations technique (the are connected to many △). Note that this graph generation has been halted at 1,000 nodes, for simplicity.

Botnet C&C servers ❶

Initial infection servers ❷

We have detected these URLs with the Distributed Hosting technique

The graph termination condition has been triggered, so we will not expand further the malicious neighborhood graph

Fig. 3. A malicious neighborhood graph, showing clients that got infected and joined a botnet (i.e., they started contacting the botnet's C&C server). The various tiny snippets of texts, which lists the URLs of all the candidates, can be ignored.

the graph, until the graph reaches its pre-set size limit (in our experiments, 4,000 nodes), or we cannot grow it anymore.

At each step, we consider if any node in the graph has some relation with entities that are not yet in the graph. If there is any, we add the corresponding entity. We consider the following relations:

- URLs belonging to a domain or FQDN (resource reuse)
- Files being downloaded from the same URL
- Domains/FQDNs being hosted on a server (resource reuse)
- URLs having the same path, or file name (availability, ease of deployment)
- Files being fetched by the same clients

For example, let's build a graph surrounding the candidate domain `04cc.com`. This domain is hosted on server 58.218.198.119, hence we add this host to the graph. This server also hosts `taobao91.com`, as Figure 2 shows, so we add the latter to the graph, and so on.

Finally, we apply some post-processing heuristics that remove the parts of the graph that do not convey any useful information for the subsequent metric calculation, such as clients that are leaves in the graph with a single parent.

While building the graph, we employ a few methods to ensure that the graph covers a sizeable portion of the malicious infrastructure it represents. We do so to avoid, for example the graph expansion iteration keeps adding URL after URL to the same domain that offers many similar-looking URLs, instead of moving to neighbor domains and thus describing a larger portion of the infrastructure. In particular, we grow the graph always in the direction of maximum interest, which is toward domains that are yet to be included in the graph. Since it conveys more information that multiple clients that visit that domain also happen to browse other suspicious domains, we apply this rule first.

We also employ some techniques to keep the graph size manageable. First, when we explore other URLs that a client in the graph has visited, we only add to the graph the URLs that one (or more) of our technique has deemed suspicious. This is to avoid inserting to the graph the benign traffic generated by the user. Also, we avoid adding additional URLs belonging to popular domains (in our experiment, domains that have been visited by more than 10% of the hosts in the dataset). Note that this does not imply that popular domains cannot be present in the graph; they can still be added as the seeding candidate, selected by our techniques. In this case, we consider this an exception and do not apply this method. In fact, both the Malicious CDN and the Exploit/Download Hosts techniques have identified suspicious candidates in the popular website `phpnuke.org`. This website has been exploited at the time when our dataset was taken and was used to serve malware to unsuspecting users. Also, to avoid the graph exploding in size, we do not expand popular file names and URL paths (such as index.html and '/').

B. Malicious-likelihood Metric

Once one of these graphs is generated for a particular suspicious candidate, simply by looking at it, a security analyst can decide with high accuracy which domains and URLs in the graphs are malicious. The rule of thumb is the following: if many suspicious candidates are included in the same graph, they are likely to be malicious. If they are very scarce, they are likely to be false positives. This, in fact, follows from the reasoning made in the previous section: both malicious and benign services have affinities with similar services. To encapsulate this guideline in a metric that can be computed automatically, we devise our malicious-likelihood metric.

In particular, our metric indicates the likelihood that the candidate under scrutiny is malicious, if all the other candidates in the graph are postulated as malicious. For each suspicious candidate, we compute the shortest distances to all the remaining suspicious candidates in the graph. Different link types, generated by the various affinities described above, carry a different link weight, depending on the likelihood that one end of the link being malicious implies that the other end is also malicious. For example, if two candidate URLs have links to the same payload (which means that they both delivered it at some point in time), and one is malicious, the other is also surely malicious. The links' weights are assigned as follows. A smaller link weight represents a higher probability that the maliciousness of one end of the link implies the maliciousness of the other:

- url ⟷ payload: weight 1
- url ⟷ server: weight 1
- url ⟷ client: weight 4
- weight 2 for any other case.

We evaluated the sensitivity that the metric has to these values: as long as the ordering of the link values is respected, the resulting values for the metric are scaled and yield similar results in the suspicious candidates rankings. Once these shortest path distances are computed, our metric M_j for the candidate j can be computed as:

$$M_j = \sum_{i \in \text{candidates}} \frac{1}{\text{shortest\_path}(i,j)}.$$

This formula encloses the concept that suspicious candidates close to the candidate under scrutiny make it more likely that the latter is malicious. In fact, the more and the closer they are, the larger the metric, whereas a few candidates far away are of little significance.

Having computed this metric for all the candidates, we can now rank them from the most to the least likely to be malicious. In our experiments, we show the performance of a maliciousness classifier that we trained, based on this metric.

VI. Evaluation

We experimentally validate our initial hypothesis that Nazca can effectively complement antivirus software and blacklists, detecting malware that evades these legacy solutions. We first provide a description of and insights into our datasets.

A. Traffic Collection

Our data comprises nine days of traffic from a commercial ISP. We use the first two, non-consecutive, days as our training set to develop our techniques and observe malicious behaviors. The remaining seven days, collected two months later, are used exclusively as our testing set. The network traces cover the traffic of residential and small business clients, using ADSL and fiber.

More specifically, the traffic for the two days in the training dataset was collected by the ISP on April 17th, 2012 and August 25th, 2012. For each TCP connection, at most 10 packets or 10 kilobytes were collected, starting from the beginning of the stream. The seven day testing dataset was collected from October 22nd to October 28th, 2012. To further limit the size of the collected data, for this dataset, at most five packets or five kilobytes were collected from every TCP connection. Moreover, the traffic to the most popular 100 domains (in volume of traffic) were removed.

In our two-days training dataset, we found 52,632 unique hosts, as identified by their IP address. 43,920 of them operate as servers, whereas 8,813 operate as clients. 101 hosts show both client and server behavior; these hosts have small websites running on a machine that is also used as a workstation. The dataset comprises of 4,431,472 HTTP conversations where the server responds with a non-whitelisted MIME type. For these connections, we counted 8,813 clients that contacted 22,232 distinct second-level domains. The seven days worth of testing dataset contains 58,335 hosts, which produced 9,037,419 HTTP requests for non-whitelisted file MIME types, to a total of 28,194 unique second-level domains. A total of 3,618,342 unique files where downloaded from 756,597 distinct URLs.

Blacklist	Records Type	Size		Present in Dataset		Present Executables		Present Malware	
		Train	Test	Train	Test	Train	Test	Train	Test
Google Safe Browsing*	URLs	-	-	116	237	22	50	22	1
Lastline	IPs and FQDNs	20,844	21,916	25	30	24	25	13	2
DNS-BH	IPs and FQDNs	23,086	25,342	86	34	76	11	3	2
PhishTank*	FQDNs	3,333	10,438	10	0	6	0	0	0
ZeusTracker*	IPs and FQDNs	1,043	1,019	3	3	2	1	0	0
Ciarmy*	IPs	100	100	0	0	0	0	0	0
Emerging Threats (Open)	IPs and FQDNs	1,848	1,927	0	2	0	2	0	1
All blacklists combined				236	271	128	77	33	4

TABLE I. MALWARE DETECTION IN OUR TRAINING AND TESTING DATASETS USING POPULAR BLACKLISTS.

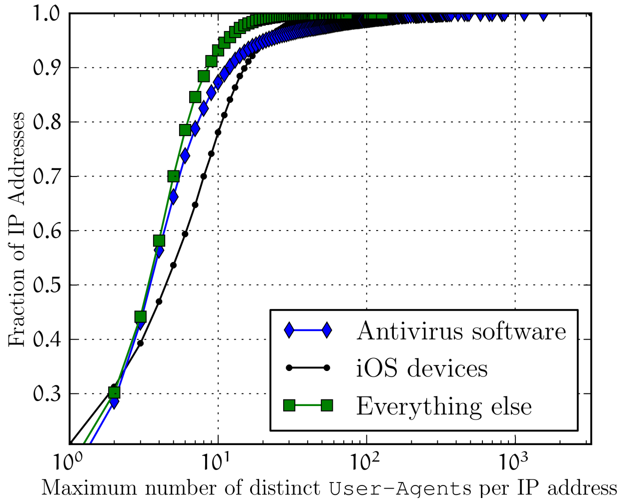


Fig. 4. CDF of the number of `User-Agents` from each IP address.

The HTTP requests that download non-whitelisted files are of our interest and form the input to our detection techniques.

To estimate the number of users served by this ISP, we look at the distribution of the `User-Agents` HTTP headers (see Figure 4). In both dataset combined, 89.9% of the IP addresses that operate exclusively as clients are associated with no more than ten different `User-Agents`. Here, we ignore `User-Agents` from popular AV software, since they keep their update version in that string. We also discard `User-Agents` from iPhones, since they encode the application name (and hence, an iPhone produces requests with a variety of `User-Agents`). Since a single workstation typically employs a few `User-Agents` (e.g., one or two browsers, several software updaters), these IP addresses are probably in use by single households, and the remaining 10.1% by small businesses.

B. *Defining Maliciousness*

Before discussing our experimental results, we need to have an operational definition of what constitutes maliciousness for a downloaded executable. In our experiments, we used VirusTotal as a basic oracle to discriminate malicious and benign executables. More precisely, we consider any software to be malicious when it tests positive against two or more AV engines run in VirusTotal. We only accept AV classifications that identify the malware strain and discard generic heuristics such as `Heuristic.LooksLike.HTML.Infected` and `IFrame.gen`. We are aware that this choice might seem counter-intuitive, since we claim that our system can detect zero-day malware. However, in the wild, many (most) samples are detected by at least some AV programs, especially after some time has passed. Thus, using VirusTotal as an approximation of the ground truth seems reasonable. Also, when we detect samples that are later confirmed by one or more antivirus engines, we do so without looking at the actual file itself.

For cases that (i) are not clearly classified by VirusTotal or (ii) match a blacklist record and test benign in VirusTotal, we run the executables in the Anubis sandbox [7] and perform manual inspection, We rely exclusively on VirusTotal for our testing dataset because of its large amount of executables. This makes deeper (manual) analysis prohibitively expensive.

For privacy reasons, after recording the traffic, the ISP has kept the dataset for several months on hold before releasing them to us. This is because the datasets contain information that should be kept private, including session cookies of the ISP's clients. Since both dataset contain only the first few kilobytes of each connection, we cannot recover the executables downloaded in the traffic directly from our data. Instead, we fetched these files from the web using the recorded URIs and discarded all those whose beginning did not match the ones present in the collected traffic. This has led us to classify executables in three categories: benign, malicious, and unknown, when we could not find a live copy of the executable. We are aware that this operational definition of maliciousness has limitations that might lead to a small amount of misclassifications, but it is the price we pay to have access to such a sensitive dataset.

C. *Blacklists*

We apply a comprehensive set of popular blacklists to our dataset to show how effectively a meticulous ISP could block malware, using the most straightforward and popular solution. The results are shown in Table I. For accuracy, we applied the version of these blacklists that was available during the dataset collection. For the blacklists that were not available during the data collections, a later version was applied and marked with an esterisk in the table.

D. *Candidate Selection Step*

To evaluate the efficacy of Nazca's Candidate Selection Step, we apply our four techniques to our training and testing datasets. When some parameters need tuning, we use (part of) the training dataset. We evaluate each technique in detail.

Before that, we show the overall results of the Candidate Selection Step in Table II. We note that the blacklists and our techniques identify two almost separate sets of malware in our datasets. Their intersection is limited to nine elements in the training dataset and none in the testing dataset.

We manually inspected the 24 downloads of malware that were detected by the blacklists but ignored by Nazca in the testing dataset. These are all pieces of malware (.doc, .jar, .dll, etc.) that have been downloaded by a single client, which did not show signs of infections observable in the traffic. For example, a client downloaded a malicious DLL after visiting a gaming website. After that, the client navigated Facebook for a few minutes, and made no other requests until midnight, when our traffic collection was halted. This kind of isolated cases defeat many of our techniques (e.g., a single download cannot trigger the File Mutations technique), and evade Nazca. This however, was expected; Nazca is designed to complement traditional solutions, not to replace them. Note also that the client in question was running ESET NOD32 antivirus (we can infer that from the `User-Agents` of its connection), which might have prevented the infection.

Finally, we attribute a higher count of candidates marked as Unknown in the testing dataset to the fact that we received this second dataset with a longer delay compared to the training dataset. Many servers have gone offline by the time we were fetching their payloads. Although this quick take down could be an indication that some malicious activity was taking place, we do not use this in our evaluation.

Technique	**Type**	**Malware**		**Benign**		**Unknown**	
		Train	Test	Train	Test	Train	Test
File Mutations	URLs	43	8	0	11	16	107
Distributed Hosting	FQDNs	45	155	4	17	42	238
Isolated Hosting	FQDNs	68	145	12	233	47	140
Exploit/Download Hosts	FQDNs	29	16	9	3	28	152
All techniques combined		117	324	9	258	85	637

TABLE II. EFFICACY OF OUR TECHNIQUES.

1) Detection of File Mutations: With this technique, we look for URLs that deliver different payloads over time. One such URL is shown in the malicious neighborhood graph in Figure 5. We have already shown a graph containing candidates detected by this technique in Figure 3 where all the C&C endpoints at the center of the figure exhibit this behavior. Applying this technique to the testing set, we also found a different set of URLs that was operating as a C&C.

In Figure 6, we show the number of variations across all URLs that show this behavior in the training dataset. If we consider only URLs that have shown more than 500 variations during these two days, we obtain exclusively malicious URLs. However, just four URLs fulfill this requisite. Instead, with the minimum possible threshold, which is two versions per day, this technique detects 49 malicious URIs in our training dataset. However, it also has 6,702 false positives, 3,864 of which are Antivirus URIs. The rest includes many failed/truncated downloads. Clearly, this performance is not acceptable and, by looking at the graph, we should set a threshold somewhere between 10 and 50. Notice that the majority of the benign URLs that trigger this technique are endpoints contacted by AV updaters. We can easily filter them with a small whitelist: specifically, only six antivirus domains account for all the false positives in this area.

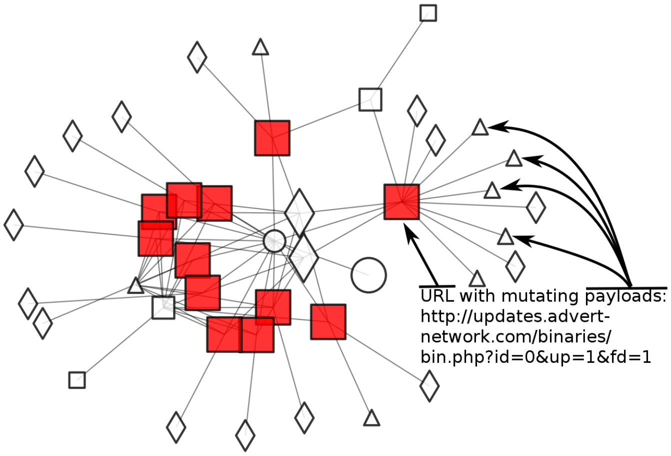

URL with mutating payloads:
http://updates.advert-network.com/binaries/bin.php?id=0&up=1&fd=1

Fig. 5. Candidate selected by the Mutating Payloads technique (for clarity, not all payloads are shown in the graph).

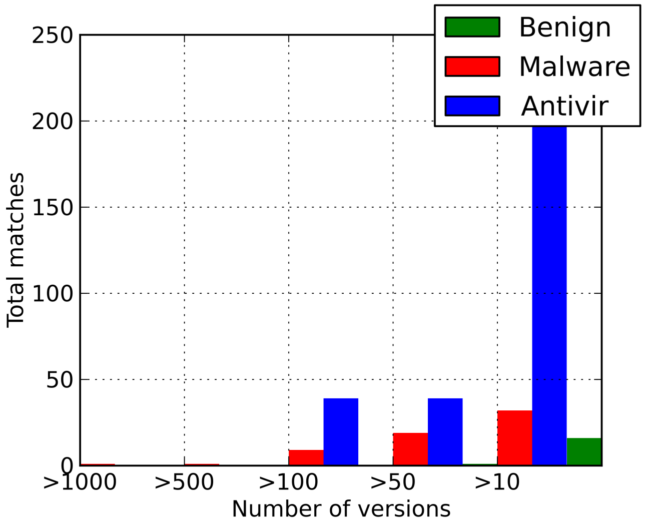


Fig. 6. Yield of different thresholds in the Mutating Payloads technique.

To set a more sensible threshold, we trained a simple classifier using five-fold cross validation on the training set, which has chosen a threshold of 12 variations a day. We show the performance of varying this threshold on the training dataset in Figure 7. The classifier choice is intuitively correct, as it yields a 91.1% true positive rate, with a 5.8% false positive rate.

We also studied the timing of the changes in the payloads, but found that there is no distinctive difference among the trends of these timings in malicious and benign URLs, other than the fact that malicious URLs tend to change more often.

2) Detection of Distributed Hosting and Domain Fluxing: We look for groups of domains that host a similar set of malicious executables. To evaluate this technique, we first discover all groups of domains that have an intersection of the file they offer. From each of these tentative distributed hosting infrastructures, we extract the features listed in Section IV-B.

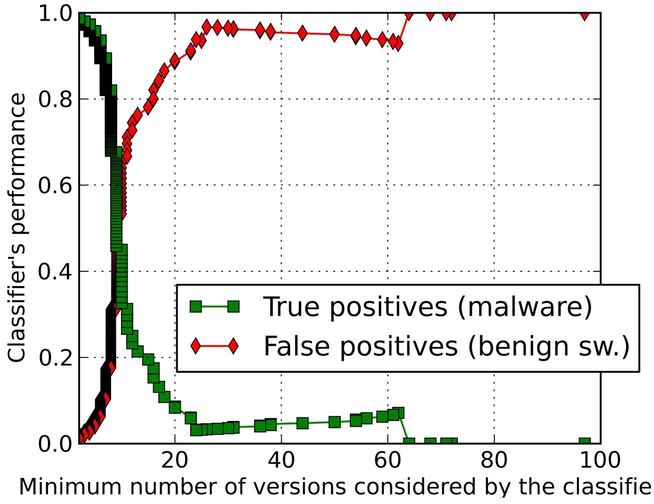


Fig. 7. Mutating Payloads technique: performance with respect to the threshold.

Distributed hosting infrastructures	Malicious	Benign	Class Precision
Predicted Malicious	12	5	70.59%
Predicted Benign	0	128	100%
Class recall	100 %	96.24%	

TABLE III. DISTRIBUTED HOSTING: CLASSIFIER PERFORMANCE.

We then train a few classifiers using the leave-one-out cross validation. The classifier that performs the best in the training dataset is a Decision Tree classifier, as shown in Figure 8. This classifier learns that a malicious distributed hosting infrastructure (i) hosts primarily executables, (ii) spans across many first-level domains, many of which are co-located on the same server, or (iii) spans across to just two or three domains, exclusively hosting the malware and a couple of supporting files (usually JS and HTML).

The classifier's performance, evaluated on the training set via the leave-one-out cross validation, is shown in Table III. Note that the classifier operates on the entire distributed hosting infrastructure as a single sample. 12 malicious infrastructures in the training set account for a total of 45 malicious domains. Three of the five benign CDNs that are incorrectly predicted as malicious belong to antivirus companies (Avira, Lavasoft, and Spybot). One of these detected malicious distributed hosting infrastructures is shown in Figure 2.

3) Detection of Dedicated Malware Hosts: We look for small domains that host a single malicious executable that is unique in the network and a small set of supporting HTML and JS files. By simply looking for domains that are involved in the download of a single unique executable file and host at most one other URL delivering HTML or JS, we obtain 241 suspicious candidates. Of these, 73 are malicious, 38 are legitimate, and 130 are either unreachable or have an expired domain record.

To boost the performance of this technique, we build a simple classifier that takes into account the registration date of the domain under analysis. If the registration has been made in the last three years (our threshold), we consider this domain as a candidate. The performance of such a classifier with respect to the threshold chosen is shown in Figure 9.

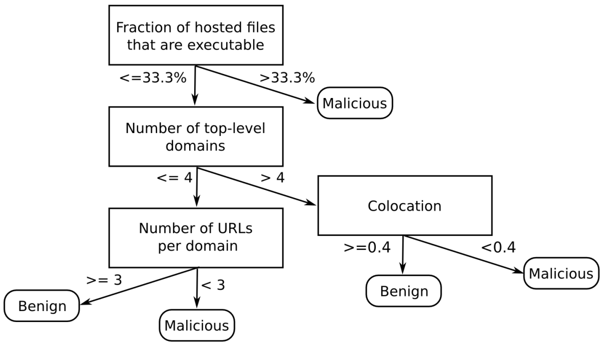

Fraction of hosted files that are executable
<=33.3%
>33.3%
Malicious
Number of top-level domains
<= 4
> 4
Colocation
>=0.4
<0.4
Benign
Malicious
Number of URLs per domain
>= 3
< 3
Benign
Malicious

Fig. 8. Distributed Hosting: decision tree classifier.

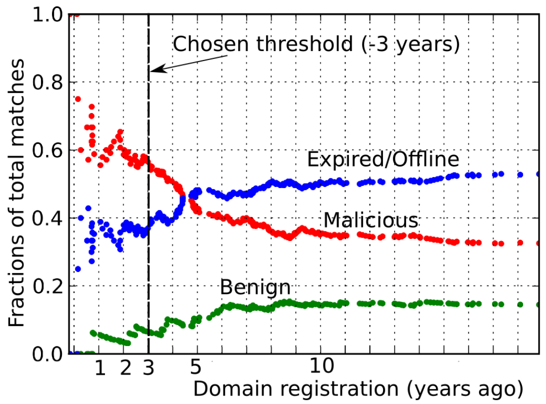


Fig. 9. Dedicated Malware Hosts technique: performance of the classifier with respect to the registration date threshold.

Figure 10 shows an example of a domain detected by this technique. `3322.org` is a Chinese dynamic DNS provider used by the Nitol botnet, which has been recently seized by Microsoft [5].

4) Detection of Exploit/Download Hosts: We look for domains that successfully perform the initial client infection. To detect them, we search for traffic that matches our model of the infection process, where a client makes a first request to the suspicious domain, followed by another request, with a different `User-Agent`, that downloads an executable. Just by selecting domains that have been contacted multiple times by the same host with different `User-Agents`, and at least one of the subsequent requests has downloaded an executable (with `User-Agent` u), we obtain 35 malicious domains and 697 legitimate ones in the training dataset.

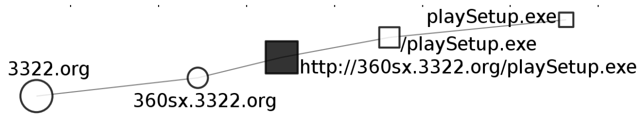


Fig. 10. Candidate selected by the Dedicated Malware Hosts technique.

To improve the detection, we train a simple classifier using as the only feature the fraction of connections, scattered in the whole dataset, performed with `User-Agent` u that resulted in downloading executables. Choosing a threshold score for the classifier at 0.9 (that is, 90% of the HTTP requests made with u in the dataset have delivered executables), we obtain a classifier that identifies 29 malicious domains and nine false positives in the training dataset.

E. Detection Step

The candidate selection step produced 1,637 unique URLs as candidates in the training dataset, and 2,581 in the testing dataset. The astute reader might notice that these numbers are different from the values listed in Table II. The reason is that the detection step operates on URIs. Table II shows the number of candidates for unique FQDNs. Since a domain can host multiple (sometimes many) distinct URIs, the numbers for URIs are larger than for FQDNs.

Using all URIs as inputs, we generate the Malicious Neighborhood graphs. Since multiple candidates might appear together in the same graph, we generated a total of 209 graphs for the training dataset and 156 graphs for the testing dataset. For the training dataset, we have 169 singleton graphs, while we have 98 for the testing dataset. The distribution of the graph sizes is shown in Figure 11. Overall, we find that the vast majority of candidates are part of a graph of non-trivial size. Note that the testing dataset graph sizes seem to be clustered. That is because we have some vast malicious neighborhoods that are only partially represented in a graph. Hence we need multiple graphs, built around different candidates belonging to these neighborhoods, to cover them completely.

We have already shown sections of these graphs throughout the paper. With them, we captured several infection-C&C Botnet infrastructures, users being tricked to download software that quietly downloads and installs malware, and a popular domain `phpnuke.org` being used to host malware. Interestingly, we have seen that a major contributor to the availability of malware comes from the ISP's caching servers; in four caching servers operated by the ISP, we found 54 malware samples. In many cases, malicious server that originally delivered them has since gone offline, but the malware was still being downloaded from the caching servers.

In addition to observing malware distribution campaigns, we used our graphs to compute the malicious-likelihood scores of all the selected candidates. With these, we have trained a simple classifier to tell apart malicious candidates from innocuous ones, using five-fold cross validation. In the training set, this classifier yields 77.35% precision and 65.77% recall on the malicious class, and 95.70% precision and 97.53% recall on the benign class. In the testing set, we have 59.81% precision and 90.14% recall on the malicious class, and 99.69% precision and 98.14% recall on the benign class. Note that some misclassifications happen because our data has uncovered an insufficient section of the malicious infrastructure and we could not see multiple candidates belonging to it. However, the results show that benign URIs are rarely mistaken as malicious (i.e., a very low false positive rate).

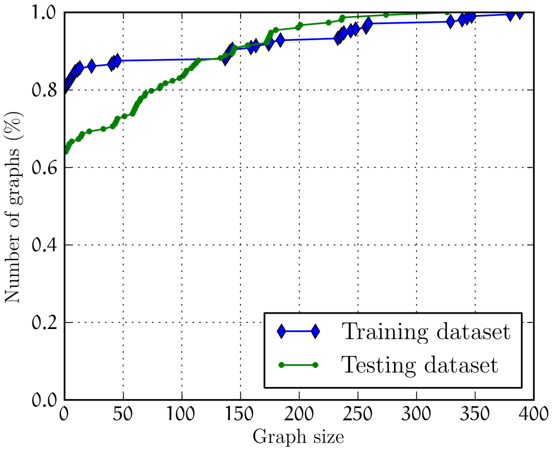


Fig. 11. Distribution of graph sizes in the Detection Step.

To better exemplify the nature of Nazca's false negatives, let's consider one of them, http://downloads.shopperreports.com/downloads/csupgrade/ComparisonShoppingUpgrade.exe. This file was downloaded only once in our dataset by a user that did not come into contact with any other malware (probably because the infection did not happen). Because of this, Nazca did not collect enough information on the infrastructure delivering it to classify this file as malicious, although it was first deemed suspicious by our Dedicated Malware Host detection technique in the second step. Nazca was actually tracking the main infrastructure orchestrating this infection campaign (the ClickPotato campaign) and the clients infected by it. This misclassificiation is then caused by the scale of the monitored network, which is not large enough for Nazca to passively observe all the various components of this large infection. To mitigate this, Nazca could be coupled with an active system, such as `EvilSeed` [8]or Li's "Finding the Linchpins" work [9], to better explore detected infrastructures and improve coverage.

In our experiments, our system detected malware samples from 19 domains that were not in our blacklists nor identified by VirusTotal. We notified antivirus companies, and within a few weeks these samples were flagged as malicious. For example, when Nazca detected the infrastructure in Figure 2, three of the payloads from URL #4 were unlisted. Being a content-agnostic system, Nazca cannot be evaded by malware that is particularly resilient resilient content analysis. Instead, to evade Nazca malware writers have to devise new, stealthier ways to distribute malware, which we envision to be challenging without sacrificing the scale of the malware distribution. Content-agnostic systems like Nazca (or [10], [11], [9]) can be evaded if malware writers deliver each piece of malware without reusing any previously-used component of their infrastructure (domain names, servers...), which will have an impact on their profits.

VII. Discussion

We discuss several design choices, usage scenarios, and evasions against Nazca.

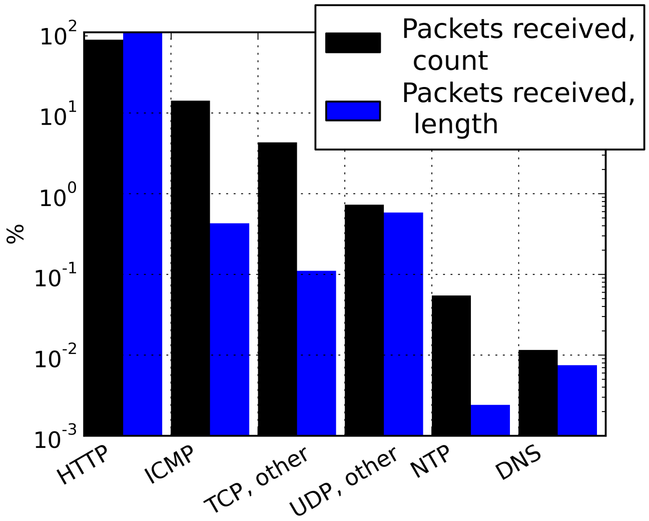


Fig. 12. Most popular protocols employed by 3,000 malware samples, executed in the Anubis sandbox

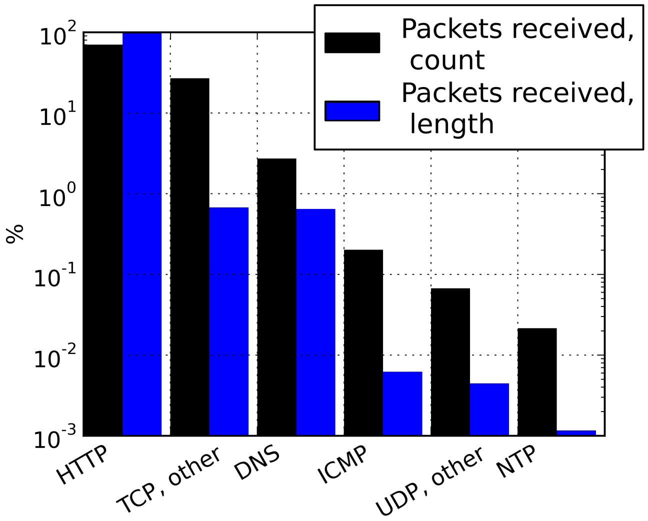


Fig. 13. Most popular protocols employed by 621 malware samples, executed in the Capture-HPC sandbox.

A. Why Only HTTP?

During our initial research, we asked ourselves what protocols we should focus on. To get an answer, we studied the traffic collected from malicious samples that were executed in two different sandboxes. The first sandbox is Anubis [7], which runs executable files in an instrumented Microsoft Windows environment. We obtained traffic of 3,000 malicious samples run by the Anubis network, and classified the network connections types. The results are shown in Figure 12. We note that HTTPS connections account for less than 1% of the total. Because we do not observe the initial exploitation in Anubis network traffic (as the malware is already on the sandbox), in this experiment, we are focusing on the control phase of the malware's lifecycle.

The second sandbox is Capture-HPC [12]. Different from Anubis, Capture-HPC is a honeyclient; it drives a web browser to visit a target domain, and gets exploited if the target delivers malware to which the honeyclient is vulnerable. We have collected the traffic for 621 infections and classified the connection types. From Figure 13, we see that even during the exploitation and installation phases of the malware's lifecycle, the protocol that overwhelmingly dominates is HTTP. Therefore, we chose to focus our efforts on HTTP traffic.

B. What If Cyber-criminals Switch to HTTPS?

The use of an encrypted channel would severely limit the data that a system like Nazca could mine, surely hampering its detection performance. However, that is true for the majority of systems that identify malware in network traffic, for example on a corporate firewall. So why do cyber-criminals still massively use HTTP? We believe that this is because switching to HTTPS would ultimately harm the cyber-criminals' revenues. There are two ways in which cyber-criminals could handle an HTTP malware server: by using a self-signed certificate or adopting a certificate signed by some trusted certification authority. On the one hand, the first solution would diminish the rate of successful infections, as clients that visit the exploit domain would be warned by their browsers that something is suspicious (all major browsers do this), and many would turn away. On the other hand, the second solution requires the cyber-criminal to obtain a valid certificate. While this might certainly happen sporadically, it is not in the interest of the certification authority to make a habit of signing certificates for exploit domains, as most major browsers would quickly remove their trust to that authority, thus hurting the certification authority business.

Another possibility is to piggyback on legitimate web services to host the cyber-criminal's software and control channels. There have been strands of malware that have used Google Drive [13], Dropbox [14], and other web services as C&C server and infection host. Nazca would not detect such malware distribution frameworks. However, the companies running these web services usually monitor their servers looking for abuse, and shut them down quickly once these abuses are discovered.

C. How Can a Cyber-criminal Evade Nazca?

There are several evasions a cyber-criminal might attempt to escape Nazca detection. One is using an encrypted protocol. We already discussed the advantages and drawback of HTTPS. Using a custom encryption protocol is possible, but such a channel might not be allowed through corporate firewalls. Therefore it is inconvenient for the cyber-criminals as it limits the basin of potential victims. A smarter solution would be piggybacking on popular software channels, such as Skype. This would defeat a system like Nazca, but it has a limited victim pool. Moreover, it is in the interest of the service provider to detect and prevent misuse of their service, as giving a free pass to malicious activities hurts their customers and ultimately their business.

Another way to reduce the chance to be detected by Nazca is to keep very small, independent malicious infrastructures. In this case, while our detection step would be ineffective, a technique like the Dedicated Malware Hosts Detection should identify these structures.

The best evasion technique that we envision is to keep on creating new, independent malware infrastructures, quickly rotating through them. Nazca needs to process several connections to the malware infrastructure before being able to identify it (although this number is fairly small; we have detected infrastructures with five connections). If the cyber-criminal has moved on to a completely different infrastructure by the time that Nazca can detect it, he will avoid the negative consequences of detection (e.g., blocking connections to his domains). However, the two infrastructures have to be completely independent; i.e., different domains, hosts, URL structures and files. Moreover, the clients that the new infrastructure contacts have to be new. Otherwise, after a couple of iterations of this infrastructure hopping, the infected clients would become very noticeable in our graph, and Nazca would flag connections to previously-unseen domains from these clients. This is quite challenging for the cyber-criminal, and it increases his operational cost, while decreasing the average timespan and number of infections.

Another evasion technique is to skip being processed by Nazca altogether, by distributing malware disguised as a multimedia file. We have seen simple cases of this kind of cloaking, where malware was distributed with a GIF image header (Nazca, running without whitelisting, detected this attack as they were mutating the images). More advanced cases might also use steganography to improve stealthiness. If the popularity of these attacks increases, Nazca will need to drop its prefilter on images and accept a performance loss and a probable increase in false detections. Ultimately, steganalysis detectors might become a necessity to combat this trend.

D. What is the Minimum Scale of the Network on Which Nazca Can Operate?

There are two factors in play here: the timespan of observation and the number of clients in the traffic. Before being operational, Nazca must observe enough traffic to learn and tune the various thresholds. The more data is collected, the more precise Nazca is. In the training dataset, with 8,813 clients, Nazca was performing detections after observing less than six hours of traffic. In smaller networks, this learning period will be longer, inversely proportional to the number of users.

In addition to the need for this initial learning phase, to achieve successful detections Nazca must observe a few interactions with the malware distribution networks. Hence, the number of users in the traffic must be large enough to have a sufficient view over these distribution infrastructures. In our datasets, 7.17% of the clients have come into contact with distribution infrastructures. To give a reference point, we identified several distribution infrastructures using the traffic of 50 infected clients in our dataset, which have been chosen randomly. Considering the fraction of infected clients in our dataset, Nazca could operate with 700 clients. With less than that, Nazca's performance gradually degrades.

Due to the sensitive nature of ISP traffic data, it was not possible for us to obtain a similar dataset from other ISPs. We are currently developing a self-contained version of Nazca, that ISPs can deploy, so that we can have a validation of our system in diverse conditions.

E. What is Nazca Space/Time Performance?

In our experiments, we have run the Candidate Selection in real time with traffic collection, on a four-cores server with 12 GB of memory. This stage is the critical one in terms of performance, as it reduces 20-fold the number of entities (URLs/domains/hosts) to be processed by the Detection stage. Thanks to this reduction, in the Detection stage, we generate only 40/58 non-trivial graphs (i.e., graphs with more than 10 nodes) in our 2/7-days dataset. We build graphs incrementally every six hours. A graph can take up to ten minutes to build from scratch (this time includes all incremental updates). The cumulative build time of trivial graphs is under five minutes.

For every connection, Nazca keeps the source and destination IP addresses and TCP port, the URL, the HTTP `User-Agent`, and an MD5 hash of the payload and its mime type. To keep our database size in check, we perform a few optimizations, such as deduplicating strings and keeping pointers to them. To give an idea of the space requirements of Nazca, the training dataset data, containing up to 10 kilobytes of every connection, requires 470 GB of storage. Nazca's database representation of it takes 4.92 GB. Considering the number of clients, this accounts for 286 KB of data per client per day. Nazca does not need to keep more than a few days of data, as it can operate with a sliding time window of collected data. These space requirements can be further lowered using a system such as `Time Machine` [4].

F. How Can Nazca be Used?

Nazca produces as output a set of web downloads (a file and the URL it was downloaded from). This information can be used to augment blacklists and prevent additional clients from contacting the infection/C&C hosts. The network administrator will also be aware of all the clients that might have contracted the infection, and can initiate the appropriate cleanup actions.

G. Can Nazca Benefit from Third-party Blacklists?

The blacklist records could be added to the candidates selected in the Candidate Selection Step, so that we have a richer set of candidates from which to generate the malicious neighborhood graphs, giving us more insight on the malicious activities they represent.

VIII. Related Work

Malware detection To identify malicious web pages, researchers have essentially adopted three strategies: (i) visiting them with *honeyclients*, (ii) statically/dynamically analyzing their content, and (iii) studying the set of malware distribution paths leading to their exploitation backends.

With honeyclients, such as Capture-HPC [15] and PhoneyC [16], researchers have visited these web pages with vulnerable browsers in virtual machines, looking for signs of infection in the guest system. This approach has very low false positives, but does not scale as each page has to be tested with an exploding variety of browser configurations [17]. Moreover, this approach is vulnerable to fingerprinting and evasions [1], yielding a high number of false negatives.

A more efficient solution is to perform some form of content analysis of the page, recognizing patterns known to be malicious [18], [19], or perform static [20] or dynamic [21] analysis of the JavaScript code. All these solutions are getting diminishing returns, as cyber-criminals thicken their code obfuscation and perfect their ability to fingerprint the analysis platforms. Researchers hence are now considering these evolving evasions against their systems as a sign of maliciousness [22].

Malware distribution infrastructures To overcome the shortcomings of the previous solutions, researchers have been invested in studying [23] and detecting malicious paths as a whole, from the initial landing pages to the infection page. Previous works have studied the infrastructures and economies of malvertising [24], Search-Engine Optimization [25], and spam [26]. Infections paths targeting surfing crowds have been passively detected analyzing the redirection chains in network traffic [11], or actively through honeyclients [27], [10]. Other researchers have proposed reputation-based systems to detect malicious downloads, such as in POLONIUM [28], where belief propagation in a tera-scale graph is used to compute the reputation of billions of downloaded executables from an initial seed of known benign and malicious files, and in CAMP [29], where the same objective is achieved inside the browser with minimal network requests. Nazca differs from all these approaches because it generates graphs containing all the heterogeneous entities involved in the malware distribution network instead of focusing on a particular class (e.g., files in CAMP and POLONIUM), thus giving a more complete view of the attackers' systems and the lifecycle of the infected hosts. For example, none of these systems could detect the totality of the complex and heterogeneous infection network shown in Figure 3.

Once an entry point to these malicious network infrastructures has been found, it can be used to further explore these networks, discovering their structure and interconnections, and thus detecting more malware sources. Researchers have done so by crawling in WEBCOP [30] and by exploiting the search engines' indexes to reach further into these networks to obtain a more complete view in EVILSEED [8].

A recent in-depth study of dedicated malicious infrastructures by Zhou [9] explores the malicious neighborhoods of an initial set of dedicated malware-spreading hosts through crawling. It uses graph mining to classify Hostname-IP clusters as topologically-dedicated hosts. Like Polonium and Nazca, it is based on the observation that there is a higher density of interconnections among malicious infrastructures than with the rest of the web. Nazca, in contrast to Zhou's crawlers, passively inspects users' activity and hence cyber-criminals' countermeasures such as cloaking, domains takedowns and parking (when run on the live traffic) are not effective against it. Moreover, Nazca does not need an initial seed of malicious hosts to expand from. We speculate that Zhou's work, being actively crawling, can gain a more comprehensive view of those infections that Nazca only sees sporadically. Finally, Nazca can monitor and correlate the whole infection process, including long-running infections on live clients, which gives Nazca an insight also on the Command & Control servers that the malware is in contact with. Given the pros and cons, we speculate that a collaboration between the two systems would give the best insight in malware distribution networks, with Nazca providing the malicious seed to bootstrap Zhou's expansion process.

IX. CONCLUSION

We analyzed how successful drive-by download exploits download and install malware programs. In particular, we developed Nazca, a system that monitors network traffic and distinguishes between downloads of legitimate and malicious programs. To make this distinction, the system observes traffic from many clients in a large-scale network. We use a number techniques to identify a set of suspicious downloads. More precisely, we look at instances where downloads exhibit behaviors that are typically associated with malicious activity that attempts to avoid traditional defenses. We then aggregate suspicious connections into a malicious neighborhood graph. This graph puts activity into context and allows us to focus on malicious entities that appear related (and hence, close together). This approach removes false positives and also paints a better picture of ongoing malware distribution campaigns. We evaluated our system on nine days of ISP traffic, during which Nazca detected almost ten million file downloads.

ACKNOWLEDGMENTS

This work was supported in part by the Office of Naval Research (ONR) under grant N000140911042, the National Science Foundation (NSF) under grants CNS-0905537 and CNS-0845559, and Secure Business Austria.

REFERENCES

[1] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna, "Escape from monkey island: Evading high-interaction honeyclients," in *IEEE Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2011, pp. 124–143.

[2] iMPERVA, "Assessing the effectiveness of antivirus solutions," Tech. Rep.

[3] A. Gostev, "The darker side of online virus scanners," http://www.securelist.com/en/weblog?weblogid=208187473.

[4] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer, "Building a time machine for efficient recording and retrieval of high-volume network traffic," in *ACM Internet Measurement Conference (IMC)*, 2005.

[5] Microsoft, "Pre-installed malware in production lines spurs microsoft's 3322.org takedown," http://www.infosecurity-magazine.com/view/28215/.

[6] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring pay-per-install: The commoditization of malware distribution." in *USENIX Security Symposium*, 2011.

[7] I. S. S. Lab, "Anubis: Analyzing unknown binaries," http://anubis.iseclab.org/.

[8] L. Invernizzi, P. M. Comparetti, S. Benvenuti, C. Kruegel, M. Cova, and G. Vigna, "Evilseed: A guided approach to finding malicious web pages," in *IEEE Symposium on Security and Privacy*, 2012.

[9] Z. Li, S. Alrwais, Y. Xie, F. Yu, M. S. Valley, and X. Wang, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," in *IEEE Symposium on Security and Privacy*, 2013.

[10] S. Lee and J. Kim, "Warningbird: Detecting suspicious urls in twitter stream," in *IEEE Symposium on Network and Distributed System Security (NDSS)*, 2012.

[11] G. Stringhini, C. Kruegel, and G. Vigna, "Shady paths: Leveraging surfing crowds to detect malicious web pages," in *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[12] C. Seifert and R. Steenson, "Capture-honeypot client (capture-hpc)," https://projects.honeynet.org/capture-hpc.

[13] S. Takashi Katsuki, “Malware targeting windows 8 uses google docs,” http://www.symantec.com/connect/blogs/malware-targeting-windows-8-uses-google-docs.

[14] C. Squared, “Killing with a borrowed knife chaining core cloud service profile infrastructure for cyber attacks,” http://www.cybersquared.com/killing-with-a-borrowed-knife-chaining-core-cloud-service-profile-infrastructure-for-cyber-attacks/.

[15] C. Seifert and R. Steenson, “Capture-honeypot client (capture-hpc),” *pp. Available at https://projects. honeynet. org/capture–hpc*, 2006.

[16] J. Nazario, “Phoneyc: A virtual client honeypot,” in *USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, 2009.

[17] Y.-M. Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and S. King, “Automated web patrol with strider honeymonkeys,” in *IEEE Symposium on Network and Distributed System Security (NDSS)*, 2006, pp. 35–49.

[18] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, “Beyond blacklists: learning to detect malicious web sites from suspicious urls,” in *ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, 2009, pp. 1245–1254.

[19] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, and M. Abadi, “deseo: Combating search-result poisoning.” in *USENIX Security Symposium*, 2011.

[20] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, “Zozzle: Low-overhead mostly static javascript malware detection,” in *USENIX Security Symposium*, 2011.

[21] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, “Rozzle: De-cloaking internet malware,” in *IEEE Symposium on Security and Privacy*, 2012, pp. 443–457.

[22] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, “Revolver: An automated approach to the detection of evasive web-based malware,” in *USENIX Security Symposium*, 2013.

[23] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, A. Pitsillidis *et al.*, “Manufacturing compromise: the emergence of exploit-as-a-service,” in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012, pp. 821–832.

[24] M. H. Moore, *Know your enemy*. Cambridge University Press, 2010.

[25] L. Lu, R. Perdisci, and W. Lee, “Surf: detecting and measuring search poisoning,” in *ACM conference on Computer and Communications Security (CCS)*, 2011, pp. 467–476.

[26] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker, “Spamscatter: Characterizing internet scam hosting infrastructure,” in *USENIX Security Symposium*, 2007.

[27] J. Zhang, C. Seifert, J. W. Stokes, and W. Lee, “ARROW: Generating signatures to detect drive-by downloads,” in *International World Wide Web Conference (WWW)*, 2011.

[28] D. H. Chau, C. Nachenberg, J. Wilhelm, A. Wright, and C. Faloutsos, “Polonium: Tera-scale graph mining and inference for malware detection,” in *SIAM International Conference on Data Mining (SDM)*, 2011.

[29] M. A. Rajab, L. Ballard, N. Lutz, P. Mavrommatis, and N. Provos, “CAMP: Content-agnostic malware protection,” in *IEEE Network and Distributed Systems Security Symposium (NDSS)*, 2013.

[30] J. W. Stokes, R. Ansersen, C. Seifert, and K. Chellapilla, “WebCop: Locating neighborhoods of malware on the web,” in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*, 2010.